Kauno technologijos universitetas

Informatikos fakultetas

Vienetų testavimas

T120B162 Programų sistemų testavimas

Antras laboratorinis darbas

Autorius
Gustas Klevinskas

Akademinė grupė
IFF-8/7

Vadovas
Doc. Šarūnas Packevičius

Kaunas, 2021

Turinys

1. Įvadas

Projektą pasirinkta testuoti xUnit testavimo įrankiu, kadangi projektas yra kuriamas C# kalba. Testinių klasių sukūrimui naudojamos AutoFixture, Moq bibliotekos. Tvarkingam rezultatų patikrinimui naudojama FluentAssertions biblioteka.

2. Testavimo „fixtures“

Testų pradinių duomenų konfigūravimui naudotas AutoFixture bei Moq bibliotekos. Šių bibliotekų pagalba yra lengvai sukuriami testinės klasės, modeliai ir pan. Su AutoFixture pagalba sukuriami AutoMapper, duombazės ir panašūs servisai, kurie perduodami į klases per DI (dependency injection).

```
public class CustomAutoDataAttribute : AutoDataAttribute {

  public CustomAutoDataAttribute() :

    base(() => new Fixture()

      .Customize(new AutoMoqCustomization { ConfigureMembers = true })

      .Customize(new EmptyEntityListCustomization())

      .Customize(new AutoMapperCustomization())

      .Customize(new DataContextCustomization())

    ) { }

}
```

Žemiau pateikta klasės iškarpa, kuri atsakinga už AutoMapper konfigūravimą testuose. Šiuo atveju jis veiks kaip ir normaliai leidžiant programą, tad geriau patikrinamas kodo tikslumas.

```
public class AutoMapperCustomization : ICustomization {

  private static readonly Lazy<IConfigurationProvider> configProvider = new(CreateAutoMapperConfig, true);

  public void Customize(IFixture fixture) {
    var provider = configProvider.Value;
    var mapper = provider.CreateMapper();

    fixture.Inject(provider);
    fixture.Inject(mapper);
  }

  private static IConfigurationProvider CreateAutoMapperConfig() {
    var services = new ServiceCollection();
    services.AddAutoMapper(typeof(Startup));

    var config = services
      .BuildServiceProvider()
      .GetRequiredService<IConfigurationProvider>();

    config.CompileMappings();
    return config;
  }
}
```

Šioje iškarpoje sukonfigūruojama duomenų bazė kompiuterio atmintyje, skirta tik testavimui. Kadangi tai nėra pilna ir tikra duombazės implementacija, nėra garantuojama, kad veikimas bus toks pat, kaip ir normaliai leidžiant programą. Naudojant tikrąją duomenų bazę, testai taptų labiau integraciniai nei vienetų.

```
public void Customize(IFixture fixture) {
  var options = new DbContextOptionsBuilder<DataContext>()
    .UseInMemoryDatabase("Test DB")
    .Options;
```

```
  var dataContext = new DataContext(options);
  var user = new User {
    Username = "Test user",
    Password = "Test password",
    Role = Role.User
  };

  dataContext.Users.Add(user);
  dataContext.SaveChanges();

  fixture.Customize<Post>(composer => composer.With(x => x.CreatedBy, user));
  fixture.Customize<Comment>(composer => composer.With(x => x.CreatedBy, user));
  fixture.Customize<CompetitionCreateModel>(composer => composer
    .With(x => x.DateFrom, DateTime.UtcNow)
    .With(x => x.DateTo, DateTime.UtcNow.Add(10.Days())));

  fixture.Inject(user);
  fixture.Inject(dataContext);
}
```

3. Testai

Sekančiuose skyreliuose bus pateikta dalis pavyzdinių testų, tikrinančių modelio laukų reikalavimus bei API servisų klases.

3.1. Modelio lauko testas

Šis testas patikrina, ar modelis teisingai indentifikuojamas kaip klaidingas paduodant įvairias gimimo metų reikšmes.

```
[Theory]
[InlineData(1799, 1)]
[InlineData(2000, 0)]
[InlineData(3000, 1)]
public void BirthYearSpecified_ExpectedResult(int birthYear, int errorCount) {
  // Arrange
  var model = new CompetitorCreateModel { BirthYear = birthYear };

  // Act
  var results = model.Validate(new ValidationContext(model));

  // Assert
  results.Should().HaveCount(errorCount);
}
```

3.2. AutoMapper konfigūracijos testas

Šis testas skirtas patikrinti AutoMapper konfigūracijai – ar visi klasės laukai yra tiesiogiai ar netiesiogiai sujungti. Tai leidžia iš anksto išvengti klaidų programos vykdymo eigoje, susijusių su kintamųjų neatsiradimu.

```
[Fact]
public void AddAutoMapperConfiguration_AssertAllConfigurationsValid() {
  // Arrange
  var services = new ServiceCollection();
  services.AddAutoMapper(typeof(Startup));

  var autoMapperConfig = services
    .BuildServiceProvider()
    .GetRequiredService<IConfigurationProvider>();

  // Act & assert
  autoMapperConfig.AssertConfigurationIsValid();
}
```

3.3. Komentarų serviso metodų testai

Žemiau pateikti kiekvieno komentarų serviso klasės metodo testai. Ganėtinai panašiai ištestuoti ir likusių servisų metodai, tad dėl glaustumo jų čia nepridėjau.

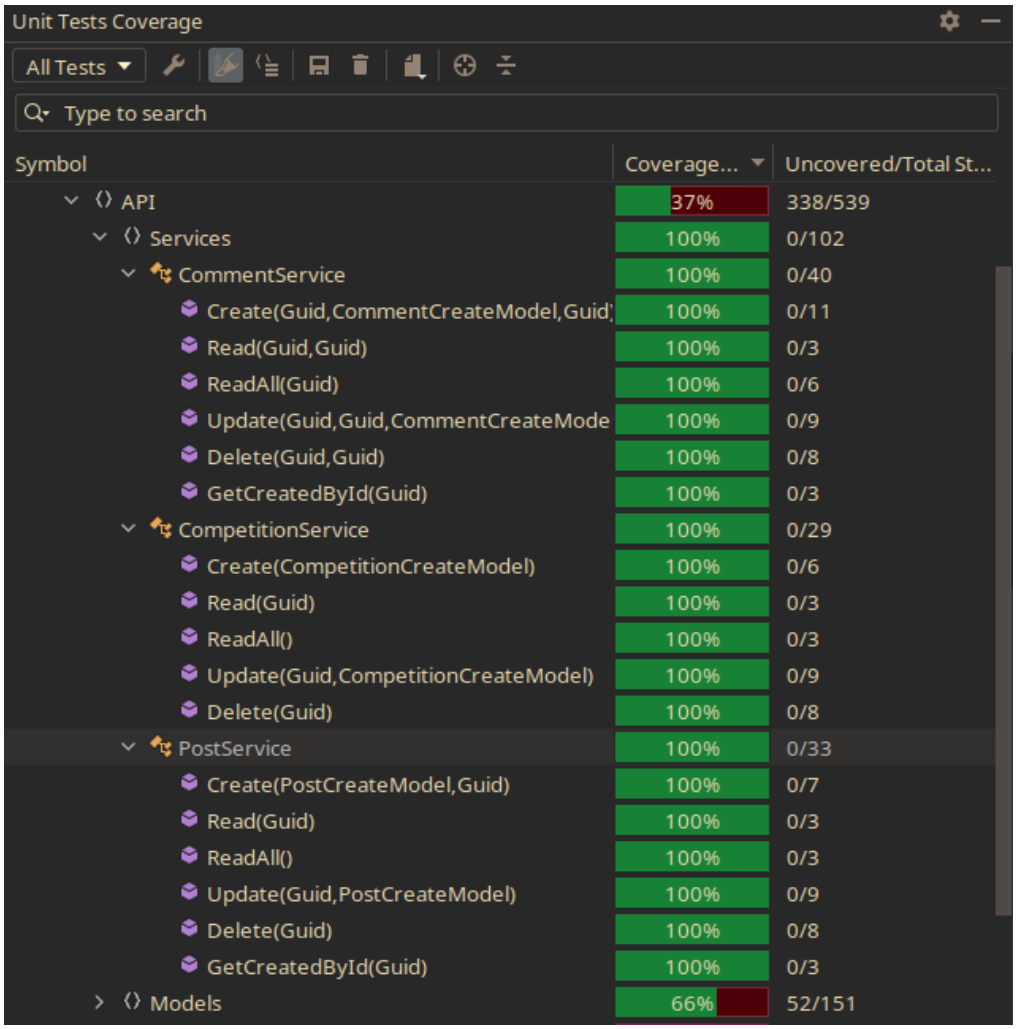
```
public class CreateTests {

  [Theory]
  [CustomAutoData]
  public void PostExists_ReturnsCreatedComment(
    DataContext dataContext,
    CommentService service,
    CommentCreateModel model,
    Post post,
    User user
  ) {
    // Arrange
    dataContext.Posts.Add(post);
    dataContext.SaveChanges();

    // Act
    var result = service.Create(post.Id, model, user.Id);

    // Assert
    result.Should().NotBeNull();
    result!.Body.Should().Be(model.Body);
    result.CreatedBy.Should().Be(user.Username);
  }

  [Theory]
  [CustomAutoData]
  public void PostDoesNotExist_ReturnsNull(
    CommentService service,
    CommentCreateModel model,
    User user
  ) {
    // Arrange

    // Act
    var result = service.Create(Guid.Empty, model, user.Id);

    // Assert
    result.Should().BeNull();
  }
}


public class ReadTests {

  [Theory]
  [CustomAutoData]
  public void CommentExists_CommentReturned(
    DataContext dataContext,
    CommentService service,
    Comment comment) {
    // Arrange
    dataContext.Comments.Add(comment);
    dataContext.SaveChanges();

    // Act
    var result = service.Read(comment.PostId, comment.Id);

    // Assert
    result.Should().NotBeNull();
    result.Should().BeEquivalentTo(new CommentReadModel {
      Id = comment.Id,
      Body = comment.Body,
      CreatedBy = comment.CreatedBy.Username,
      CreatedOn = comment.CreatedOn
    });
  }
}


public class ReadAllTests {
```

```
  [Theory]
  [CustomAutoData]
  public void PostExists_ReturnsCreatedComment(
    DataContext dataContext,
    CommentService service,
    Post post,
    Comment comment
  ) {
    // Arrange
    post.Comments.Add(comment);
    dataContext.Posts.Add(post);
    dataContext.SaveChanges();

    // Act
    var result = service.ReadAll(post.Id)?.ToList();

    // Assert
    result.Should().NotBeNull();
    result.Should()
      .HaveCount(1)
      .And
      .ContainEquivalentOf(new CommentReadModel {
        Id = comment.Id,
        Body = comment.Body,
        CreatedBy = comment.CreatedBy.Username,
        CreatedOn = comment.CreatedOn
      });
  }

  [Theory]
  [CustomAutoData]
  public void PostDoesNotExist_ReturnsNull(CommentService service) {
    // Arrange

    // Act
    var result = service.ReadAll(Guid.Empty);

    // Assert
    result.Should().BeNull();
  }
}


public class UpdateTests {

  [Theory]
  [CustomAutoData]
  public void CommentExists_ReturnsUpdatedComment(
    DataContext dataContext,
    CommentService service,
    Comment comment,
    CommentCreateModel model
  ) {
    // Arrange
    var bodyBefore = comment.Body;
    dataContext.Comments.Add(comment);
    dataContext.SaveChanges();

    // Act
    var result = service.Update(comment.PostId, comment.Id, model);

    // Assert
    result.Should().NotBeNull();
    result!.Should().NotBe(bodyBefore);
    result.Should()
      .BeEquivalentTo(new CommentReadModel {
        Id = comment.Id,
        Body = model.Body,
        CreatedBy = comment.CreatedBy.Username,
        CreatedOn = comment.CreatedOn
      });
  }

  [Theory]
  [CustomAutoData]
```

```
  public void CommentDoesNotExist_ReturnsNull(
    CommentService service,
    Comment comment,
    CommentCreateModel model) {
    // Arrange

    // Act
    var result = service.Update(comment.PostId, Guid.Empty, model);

    // Assert
    result.Should().BeNull();
  }
}


public class DeleteTests {

  [Theory]
  [CustomAutoData]
  public void CommentExists_DeletedAndReturnsTrue(
    DataContext dataContext,
    CommentService service,
    Comment comment) {
    // Arrange
    dataContext.Comments.Add(comment);
    dataContext.SaveChanges();

    // Act
    var result = service.Delete(comment.PostId, comment.Id);

    // Assert
    result.Should().BeTrue();
    dataContext.Comments.Find(comment.Id).Should().BeNull();
  }

  [Theory]
  [CustomAutoData]
  public void CommentDoesNotExist_ReturnsFalse(
    CommentService service) {
    // Arrange

    // Act
    var result = service.Delete(Guid.Empty, Guid.Empty);

    // Assert
    result.Should().BeFalse();
  }
}


public class GetCreatedByIdTests {

  [Theory]
  [CustomAutoData]
  public void CommentExists_ReturnsUserId(
    DataContext dataContext,
    CommentService service,
    Comment comment) {
    // Arrange
    dataContext.Comments.Add(comment);
    dataContext.SaveChanges();

    // Act
    var result = service.GetCreatedById(comment.Id);

    // Assert
    result.Should().Be(comment.CreatedBy.Id);
  }
}
```

4. Testų padengimas

Pav 1. pateiktas kodo padengimas vienetų testais. Visi servisų metodai yra pilnai padengti. Modelių rašo, jog nėra pilnas padengimas, tačiau tai dėl nepanaudotų setter'ių. Jų testuoti servisų testuose nėra tikslo, nes ASP.NET karkasas automatiškai nustato modelio reikšmes.



1 pav. *Testų padengimas*

5. Išvados

Pavyko sėkmingai parašyti 29 testus. Gautas 100% kodo padengimas servisų klasėse, tačiau liko mažas modelių ir kontrolerių kodo padengimas. Modeliuose liko nepratestuoti setter'iai, kurie yra automatiškai priskiriami ateinant HTTP užklausai. Dėl netikrų duomenų padavimo bei autentifikacijos, kontrolerių testai buvo neparašyti. Juos planuojama ištestuoti integraciniais testais ateityje.